E7322020

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

CSC 580

ADVANCED PYTHON PROGRAMMING

Submitted by

SURESH N – E7322020

MASTER OF SCIENCE

in

DATA ANALYTICS

Sri Ramachandra Faculty of Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai -600116

JAN,2023

BONAFIDE CERTIFICATE

Certified that this project report is the bonafide record of work done by **“SURESH N–E7322020”**.

Signature of the Course Faculty

Signature of Vice-Principal

Dr. Pitchumani Angayarkanni S

Associate Professor,

Department of Computer Science and Engineering

Sri Ramachandra Faculty of Engineering and Technology,

SRIHER, Porur, Chennai-600 116.

Prof. M. Prema

Vice-Principal,

Department of Computer Science and Engineering

Sri Ramachandra Faculty of Engineering and Technology,

SRIHER, Porur, Chennai-600 116.

Evaluation Date:

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

TABLE OF CONTENTS

1.

Data Science engineer your task is to first analyse the dataset and its features. Perform preprocessing and data normalization techniques which is a preliminary stage for an effective prediction machine learning model. CO1

Dataset: titanic dataset and cancer dataset

Perform the following on the dataset

A. Total number of observations and features.

B. Find missing values if any in the columns and replace the missing values based on relevant statistical analysis.

C. Perform SMOTE analysis to oversample if the dataset is imbalanced.

D. Rescale the data using MinMaxScaler and StandardScaler.

```
from sklearn import datasets
import pandas as pd
import numpy as np
data=pd.read_csv("C:\\python_code\\titanic - titanic.csv")
data
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

INTERPRETATION:

Total number of observations:891

data.isnull().sum()

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

INTERPRETATION

Missing values found in the given dataset in the column age, cabin, and embarked.

data=data.select_dtypes(exclude=['object'])

from sklearn.impute import SimpleImputer

imputer=SimpleImputer(strategy='most_frequent')

data.iloc[:,:]=imputer.fit_transform(data)

data.isnull().sum()

```
PassengerId    0
Survived       0
Pclass         0
Age            0
SibSp          0
Parch          0
Fare           0
dtype: int64
```

X=data.drop('Survived',axis=1)

y=data['Survived']

from imblearn.over_sampling import SMOTE

oversample=SMOTE()

X,y=oversample.fit_resample(X,y)

X.shape

```
(1098, 6)
```

y.shape

```
(1098,)
```

y.describe().T

```
count    1098.000000
mean        0.500000
std         0.500228
min         0.000000
25%         0.000000
50%         0.500000
75%         1.000000
max         1.000000
Name: Survived, dtype: float64
```

<u>INTERPRETATION</u>

The dataset is balanced using oversample method in SMOTE analysis.

#Standardization

data=data.select_dtypes(exclude=['object'])

#standardization-standardscaler

from sklearn.preprocessing import StandardScaler

#remove the id and class label columns

```
scaler=StandardScaler()
data.iloc[:,:]=scaler.fit_transform(data)
data
```

:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	-1.730108	-0.789272	0.827377	-0.497793	0.432793	-0.473674	-0.502445
1	-1.726220	1.266990	-1.566107	0.715048	0.432793	-0.473674	0.786845
2	-1.722332	1.266990	0.827377	-0.194583	-0.474545	-0.473674	-0.488854
3	-1.718444	1.266990	-1.566107	0.487640	0.432793	-0.473674	0.420730
4	-1.714556	-0.789272	0.827377	0.487640	-0.474545	-0.473674	-0.486337
...	...	...	...	...	...	...	...
886	1.714556	-0.789272	-0.369365	-0.118780	-0.474545	-0.473674	-0.386671
887	1.718444	1.266990	-1.566107	-0.725201	-0.474545	-0.473674	-0.044381
888	1.722332	-0.789272	0.827377	-0.346188	0.432793	2.008933	-0.176263
889	1.726220	1.266990	-1.566107	-0.194583	-0.474545	-0.473674	-0.044381
890	1.730108	-0.789272	0.827377	0.260233	-0.474545	-0.473674	-0.492378

891 rows × 7 columns

```
from sklearn import preprocessing
#scaler=preprocessing.MinMaxScaler()->default feature range=(0,1)
scalar=preprocessing.MinMaxScaler(feature_range=(0,2))
data.iloc[:,:]=scalar.fit_transform(data)
data
```

]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	0.000000	0.0	2.0	0.542347	0.25	0.000000	0.028302
1	0.002247	2.0	0.0	0.944458	0.25	0.000000	0.278271
2	0.004494	2.0	2.0	0.642875	0.00	0.000000	0.030937
3	0.006742	2.0	0.0	0.869063	0.25	0.000000	0.207289
4	0.008989	0.0	2.0	0.869063	0.00	0.000000	0.031425
...	...	...	...	...	...	...	...
886	1.991011	0.0	1.0	0.668007	0.00	0.000000	0.050749
887	1.993258	2.0	0.0	0.466951	0.00	0.000000	0.117112
888	1.995506	0.0	2.0	0.592611	0.25	0.666667	0.091543
889	1.997753	2.0	0.0	0.642875	0.00	0.000000	0.117112
890	2.000000	0.0	2.0	0.793667	0.00	0.000000	0.030254

891 rows × 7 columns

cancer=datasets.load_breast_cancer()

cancer

```
: {'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
          1.189e-01],
         [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
          8.902e-02],
         [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
          8.758e-02],
         ...,
         [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
          7.820e-02],
         [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
          1.240e-01],
         [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
          7.039e-02]]),
  'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
         0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
         1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
         1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
         1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
         0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
         1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
         0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
         1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
         1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
```

df_cancer = pd.DataFrame(cancer.data, columns=cancer.feature_names)

df_cancer['target'] = pd.Series(cancer.target) df_cancer

]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	com
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.16220	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.12380	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.14440	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.20980	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.13740	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.14100	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.11660	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	0.11390	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	0.16500	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	0.08996	

569 rows × 31 columns

df_cancer.isnull().sum()

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
```

INTERPRETATION

No missing values found in the given dataset.

X=df_cancer.drop('target',axis=1)

y=df_cancer['target']

from imblearn.over_sampling import SMOTE

oversample=SMOTE()

X,y=oversample.fit_resample(X,y)

X.shape

```
(714, 30)
```

y.shape

```
(714,)
```

y.describe().T

```
count    714.000000
mean       0.500000
std        0.500351
min        0.000000
25%        0.000000
50%        0.500000
75%        1.000000
max        1.000000
Name: target, dtype: float64
```

INTERPRETATION

The dataset is balanced using oversample method in SMOTE Analysis.

#standardization-standardscaler

from sklearn.preprocessing import StandardScaler

#remove the id and class label columns

scaler=StandardScaler()

df_cancer.iloc[:,:]=scaler.fit_transform(df_cancer)

df_cancer

]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	sm
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	...	-1.359293	2.303601	2.001237	
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	-0.868652	...	-0.369203	1.535126	1.890489	-
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.939685	-0.398008	...	-0.023974	1.347475	1.456285	
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	4.910919	...	0.133984	-0.249939	-0.550021	
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	-0.562450	...	-1.466770	1.338539	1.220724	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.219060	1.947285	2.320965	-0.312589	-0.931027	...	0.117700	1.752563	2.015301	
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.017833	0.693043	1.263669	-0.217664	-1.058611	...	2.047399	1.421940	1.494959	-
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.038680	0.046588	0.105777	-0.809117	-0.895587	...	1.374854	0.579001	0.427906	-
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.272144	3.296944	2.658866	2.137194	1.043695	...	2.237926	2.303601	1.653171	
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.150752	-1.114873	-1.261820	-0.820070	-0.561032	...	0.764190	-1.432735	-1.075813	-

from sklearn import preprocessing

#scaler=preprocessing.MinMaxScaler()->default feature range=(0,1)

scalar=preprocessing.MinMaxScaler(feature_range=(0,2))

df_cancer.iloc[:,:]=scalar.fit_transform(df_cancer)

df_cancer

]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	w smooth
0	1.042075	0.045316	1.091977	0.727466	1.187506	1.584075	1.406279	1.462227	1.372727	1.211036	...	0.283049	1.336620	0.901396	1.20
1	1.286289	0.545147	1.231567	1.003181	0.579760	0.363536	0.407216	0.697515	0.759596	0.282645	...	0.607143	1.079635	0.870429	0.69
2	1.202991	0.780521	1.191486	0.898834	1.028618	0.862033	0.925023	1.271372	1.019192	0.422494	...	0.720149	1.016883	0.749017	0.96
3	0.420181	0.721677	0.467003	0.205811	1.622642	1.622723	1.131209	1.045726	1.552525	2.000000	...	0.771855	0.482693	0.188016	1.83
4	1.259785	0.313155	1.261972	0.978579	0.860702	0.695786	0.927835	1.036779	0.756566	0.373631	...	0.247868	1.013895	0.683150	0.87
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	1.379999	0.857626	1.357335	1.132980	1.053895	0.592111	1.142924	1.380716	0.672727	0.264111	...	0.766525	1.152348	0.905328	0.92
565	1.244640	1.253974	1.208071	0.948038	0.815564	0.515429	0.674789	0.973260	0.698990	0.226201	...	1.398188	1.041785	0.759831	0.60
566	0.910502	1.242475	0.891576	0.606235	0.576329	0.508680	0.433505	0.527038	0.535354	0.274642	...	1.178038	0.759898	0.461463	0.56
567	1.289129	1.327021	1.331076	0.951432	1.176672	1.580394	1.646673	1.510934	1.350505	0.850885	...	1.460554	1.336620	0.804070	1.23
568	0.073738	1.003044	0.057080	0.031813	0.000000	0.148703	0.000000	0.000000	0.532323	0.374052	...	0.978145	0.087156	0.040995	0.24

569 rows × 31 columns

2. Perform the following operations on the datasets specified below:

Dataset 1: planet dataset from seaborn

Dataset 2: titanic dataset CO2

i.Load the dataset and display top 10 records and bottom 5 records

ii.Statistically analyse the overall properties of the dataset using a single command after dropping null or missing values

iii.Find the mean value of orbital periods (in days) that each method is sensitive to.

iv.Perform multiple aggregation like min, max and mean on the column orbital period.

v.Statistically analyse the overall properties of the dataset using a single command after dropping null or missing values

vi.Perform data imputation based on most frequent data value

vii. Normalize the dataset

viii. Analyse the correlation between the features

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import scipy

import seaborn as sns

#Load the dataset and display top 10 records and bottom 5 records

df=sns.load_dataset('titanic')

print(df.head(10))

print(df.tail(5))

```
   survived  pclass     sex   age  sibsp  parch     fare embarked   class  \
0         0       3    male  22.0      1      0   7.2500        S   Third
1         1       1  female  38.0      1      0  71.2833        C   First
2         1       3  female  26.0      0      0   7.9250        S   Third
3         1       1  female  35.0      1      0  53.1000        S   First
4         0       3    male  35.0      0      0   8.0500        S   Third
5         0       3    male   NaN      0      0   8.4583        Q   Third
6         0       1    male  54.0      0      0  51.8625        S   First
7         0       3    male   2.0      3      1  21.0750        S   Third
8         1       3  female  27.0      0      2  11.1333        S   Third
9         1       2  female  14.0      1      0  30.0708        C  Second

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
5    man        True  NaN   Queenstown    no   True
6    man        True    E  Southampton    no   True
7  child       False  NaN  Southampton    no  False
8  woman       False  NaN  Southampton   yes  False
9  child       False  NaN    Cherbourg   yes  False
     survived  pclass     sex   age  sibsp  parch   fare embarked   class  \
886         0       2    male  27.0      0      0  13.00        S  Second
887         1       1  female  19.0      0      0  30.00        S   First
888         0       3  female   NaN      1      2  23.45        S   Third
889         1       1    male  26.0      0      0  30.00        C   First
890         0       3    male  32.0      0      0   7.75        Q   Third
```

df.columns

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

INTERPRETATION

Missing values found in the column age and deck.

#Statistically analyse the overall properties of the dataset using a single command after dropping null or missing values

df.dropna().describe()

:

	survived	pclass	age	sibsp	parch	fare
count	182.000000	182.000000	182.000000	182.000000	182.000000	182.000000
mean	0.675824	1.192308	35.623187	0.467033	0.478022	78.919735
std	0.469357	0.516411	15.671615	0.645007	0.755869	76.490774
min	0.000000	1.000000	0.920000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	24.000000	0.000000	0.000000	29.700000
50%	1.000000	1.000000	36.000000	0.000000	0.000000	57.000000
75%	1.000000	1.000000	47.750000	1.000000	1.000000	90.000000
max	1.000000	3.000000	80.000000	3.000000	4.000000	512.329200

#Perform data imputation based on most frequent data value

from sklearn.impute import SimpleImputer

impute = SimpleImputer(strategy='most_frequent')

df.iloc[:,:] = impute.fit_transform(df)

df.isnull().sum()

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

#Normalize the dataset

from sklearn import preprocessing

df.select_dtypes(exclude=[object]).iloc[:,:]=preprocessing.normalize(df.select_dtypes(exclude=[object]))

df

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	0.940169	1	0	0.309828	S	Third	man	True	C	Southampton	no	False
1	1	1	female	0.470309	1	0	0.882241	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	0.949509	0	0	0.289418	S	Third	woman	False	C	Southampton	yes	True
3	1	1	female	0.550134	1	0	0.834632	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	0.970426	0	0	0.223198	S	Third	man	True	C	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	0.898007	0	0	0.432374	S	Second	man	True	C	Southampton	no	True
887	1	1	female	0.534417	0	0	0.843816	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	0.710849	1	2	0.694559	S	Third	woman	False	C	Southampton	no	False
889	1	1	male	0.654101	0	0	0.754732	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	0.967009	0	0	0.234197	Q	Third	man	True	C	Queenstown	no	True

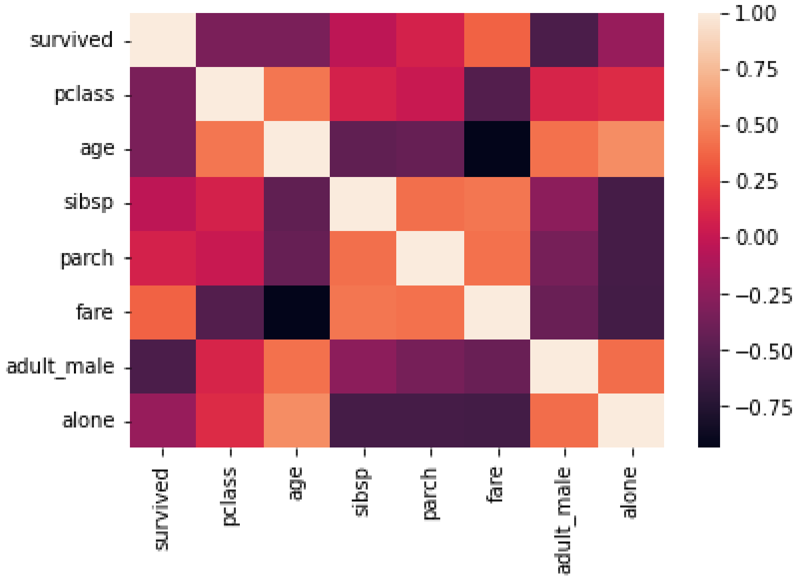
891 rows × 15 columns

#Analyse the correlation between the features

corr=df.corr()

sns.heatmap(corr)

```
<AxesSubplot:>
```



The stronger relationship shows between the age and survived.

3. Perform EDA on the Indian Premier league dataset. Formulate research questions based on the dataset and perform analysis on the same

import pandas as pd

import seaborn as sns

#load the data

df=pd.read_csv("C:\\python_code\\ipl.csv")

df

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wicke
0	1	2008	Bangalore	2008-04-18	Kolkata Knight Riders	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Kolkata Knight Riders	140	
1	2	2008	Chandigarh	2008-04-19	Chennai Super Kings	Kings XI Punjab	Chennai Super Kings	bat	normal	0	Chennai Super Kings	33	
2	3	2008	Delhi	2008-04-19	Rajasthan Royals	Delhi Daredevils	Rajasthan Royals	bat	normal	0	Delhi Daredevils	0	
3	4	2008	Mumbai	2008-04-20	Mumbai Indians	Royal Challengers Bangalore	Mumbai Indians	bat	normal	0	Royal Challengers Bangalore	0	
4	5	2008	Kolkata	2008-04-20	Deccan Chargers	Kolkata Knight Riders	Deccan Chargers	bat	normal	0	Kolkata Knight Riders	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
572	573	2016	Raipur	2016-05-22	Delhi Daredevils	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	
573	574	2016	Bangalore	2016-05-24	Gujarat Lions	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 577 entries, 0 to 576
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               577 non-null    int64
 1   season           577 non-null    int64
 2   city             570 non-null    object
 3   date             577 non-null    object
 4   team1            577 non-null    object
 5   team2            577 non-null    object
 6   toss_winner      577 non-null    object
 7   toss_decision    577 non-null    object
 8   result           577 non-null    object
 9   dl_applied       577 non-null    int64
 10  winner           574 non-null    object
 11  win_by_runs      577 non-null    int64
 12  win_by_wickets   577 non-null    int64
 13  player_of_match  574 non-null    object
 14  venue            577 non-null    object
 15  umpire1          577 non-null    object
 16  umpire2          577 non-null    object
 17  umpire3          0 non-null      float64
dtypes: float64(1), int64(5), object(12)
memory usage: 81.3+ KB
```

INTERPRETATION

Missing values found in the dataset in city,winner,umpire_3 columns.

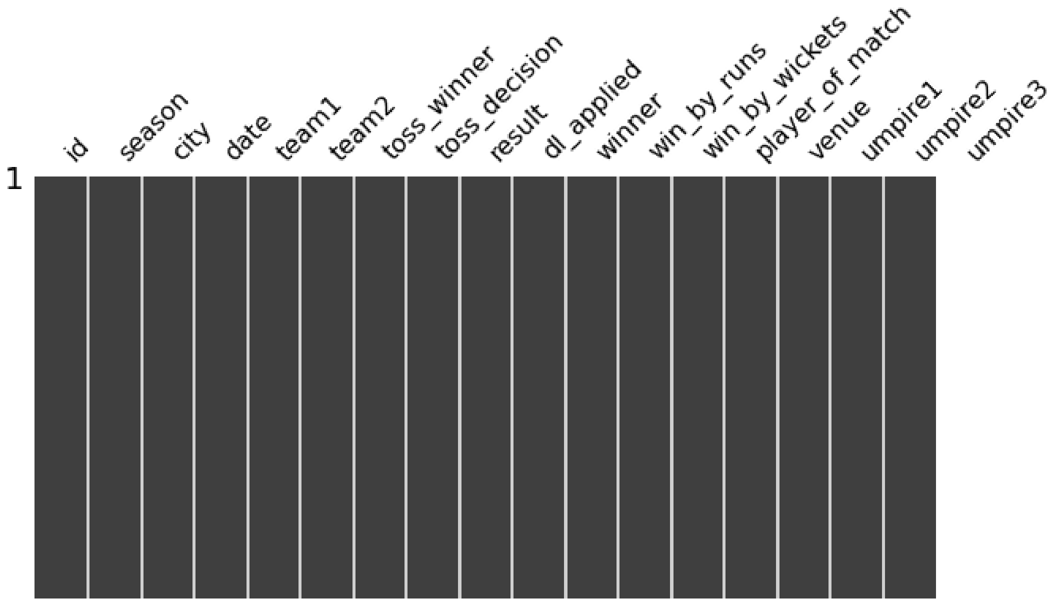
df.isnull().sum()

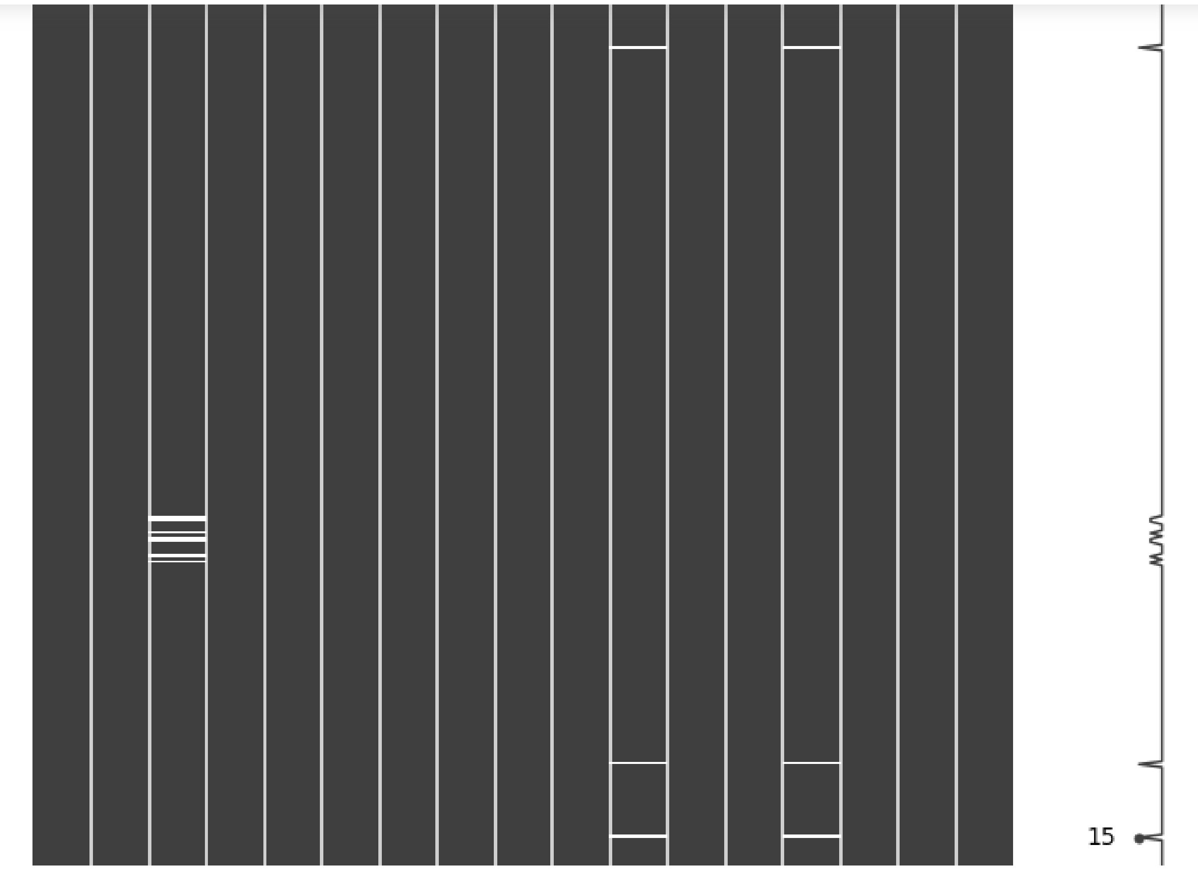
```
id                  0
season              0
city                7
date                0
team1               0
team2               0
toss_winner         0
toss_decision       0
result              0
dl_applied          0
winner              3
win_by_runs         0
win_by_wickets      0
player_of_match     3
venue               0
umpire1             0
umpire2             0
umpire3           577
dtype: int64
```

#to visualize missing value

import missingno as msno

msno.matrix(df,figsize=(12,18))





INTERPRETATION

Missing at random

df.describe()

:

	id	season	dl_applied	win_by_runs	win_by_wickets	umpire3
count	577.000000	577.000000	577.000000	577.000000	577.000000	0.0
mean	289.000000	2012.029463	0.025997	13.715771	3.363951	NaN
std	166.709828	2.486247	0.159263	23.619282	3.416049	NaN
min	1.000000	2008.000000	0.000000	0.000000	0.000000	NaN
25%	145.000000	2010.000000	0.000000	0.000000	0.000000	NaN
50%	289.000000	2012.000000	0.000000	0.000000	3.000000	NaN
75%	433.000000	2014.000000	0.000000	20.000000	6.000000	NaN
max	577.000000	2016.000000	1.000000	144.000000	10.000000	NaN

#data types

df.dtypes

```
id                  int64
season              int64
city               object
date               object
team1              object
team2              object
toss_winner        object
toss_decision      object
result             object
dl_applied          int64
winner             object
win_by_runs         int64
win_by_wickets      int64
player_of_match    object
venue              object
umpire1            object
umpire2            object
umpire3           float64
dtype: object
```

number of rows and columns

df.shape

```
(577, 18)
```

#number of dimensions

df.ndim

```
2
```

#unique values of the data

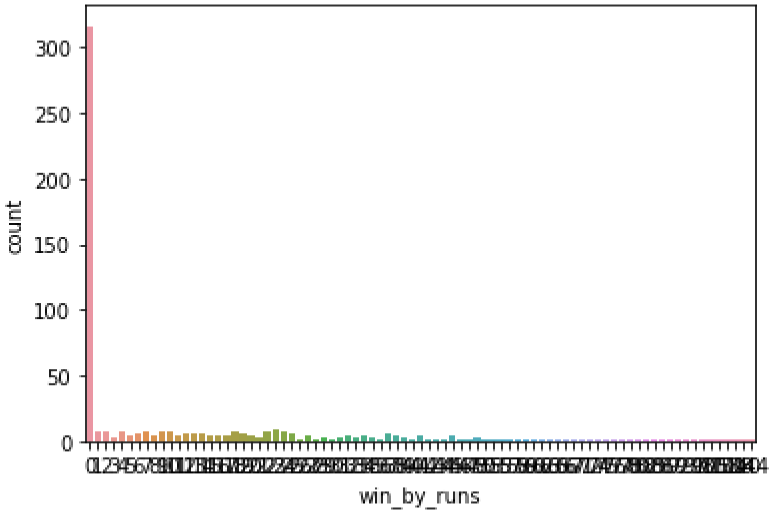
df['win_by_runs'].unique()

```
array([140,  33,   0,   6,  66,  13,  10,  45,   9,  29,   5,  18,  23,
        41,  12,  65,  25,   3,   1,  14, 105,  19,  75,  92,  11,  24,
        27,  38,   8,  78,  16,  53,   2,   4,  31,  55,  98,  34,  36,
        39,  17,  40,  67,  63,  37,  57,  35,  22,  21,  48,  26,  20,
        85,  32,  76, 111,  82,  43,  58,  28,  74,  42,  59,  46,   7,
        47,  86,  44,  87, 130,  15,  60,  77,  30,  50,  93,  72,  62,
        97, 138,  71, 144,  80], dtype=int64)
```

#to visualize the unique value

sns.countplot(df['win_by_runs'])

```
<AxesSubplot:xlabel='win_by_runs', ylabel='count'>
```



df.columns

```
Index(['id', 'season', 'city', 'date', 'team1', 'team2', 'toss_winner',
       'toss_decision', 'result', 'dl_applied', 'winner', 'win_by_runs',
       'win_by_wickets', 'player_of_match', 'venue', 'umpire1', 'umpire2',
       'umpire3'],
      dtype='object')
```

ipl1=df.drop(['umpire3','city','winner','player_of_match'], axis=1)

ipl1

ipl1.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 577 entries, 0 to 576
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   id              577 non-null    int64 
 1   season          577 non-null    int64 
 2   date            577 non-null    object
 3   team1           577 non-null    object
 4   team2           577 non-null    object
 5   toss_winner     577 non-null    object
 6   toss_decision   577 non-null    object
 7   result          577 non-null    object
 8   dl_applied      577 non-null    int64 
 9   win_by_runs     577 non-null    int64 
 10  win_by_wickets  577 non-null    int64 
 11  venue           577 non-null    object
 12  umpire1         577 non-null    object
 13  umpire2         577 non-null    object
dtypes: int64(5), object(9)
memory usage: 63.2+ KB
```

Dropping the columns containing missing values.

ipl1.isnull().sum()

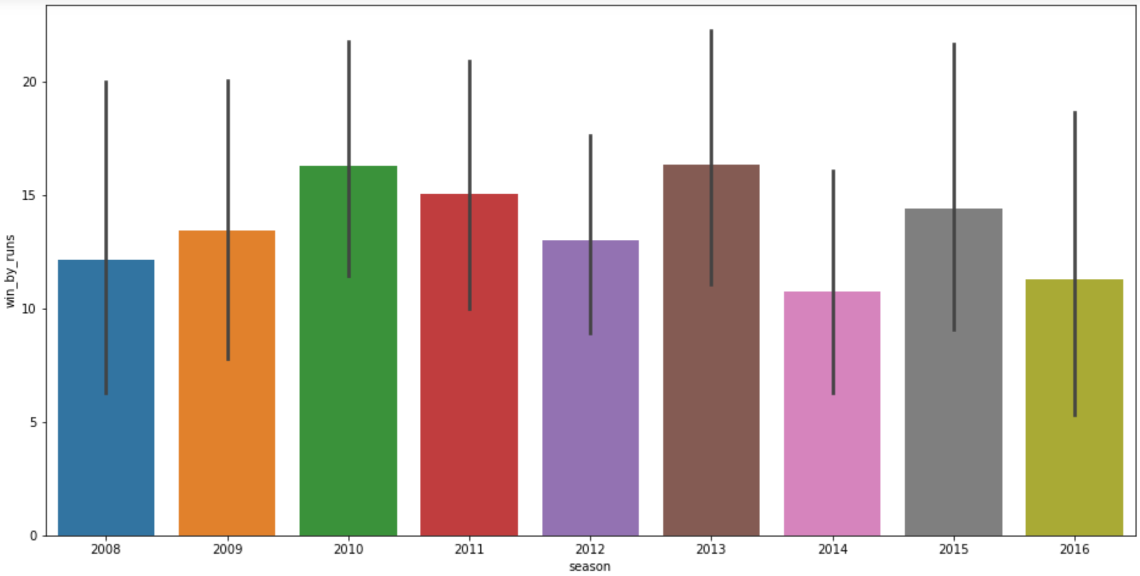
```
id                0
season            0
date              0
team1             0
team2             0
toss_winner       0
toss_decision     0
result            0
dl_applied        0
win_by_runs       0
win_by_wickets    0
venue             0
umpire1           0
umpire2           0
dtype: int64
```

ipl=ipl.groupby('season')['win_by_runs'].sum().reset_index()

print(ipl)

	season	win_by_runs
0	2008	705
1	2009	764
2	2010	976
3	2011	1098
4	2012	960
5	2013	1241
6	2014	644
7	2015	850
8	2016	676

```
import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize = (16,8))

sns.barplot(x="season", y="win_by_runs", data=ipl1)

plt.show()
```



INTERPRETATION

Barplot shows the increase in win by runs over years. Results stated that win by runs reaches its peak in the year of 2013.

```
#correlation

ipl1.corr()
```

:

	id	season	dl_applied	win_by_runs	win_by_wickets
id	1.000000	0.992806	0.017197	-0.014813	-0.012804
season	0.992806	1.000000	0.015600	-0.018098	-0.005966
dl_applied	0.017197	0.015600	1.000000	-0.005878	-0.023803
win_by_runs	-0.014813	-0.018098	-0.005878	1.000000	-0.572839
win_by_wickets	-0.012804	-0.005966	-0.023803	-0.572839	1.000000

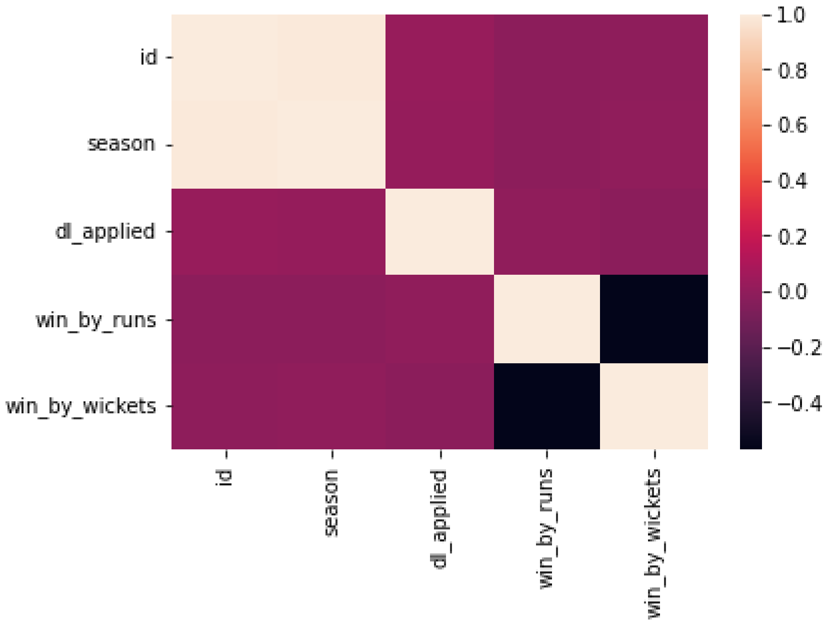
INTERPRETATION

Negative correlation between season and win_by_runs.

#correlation plot

sns.heatmap(ipl1.corr())

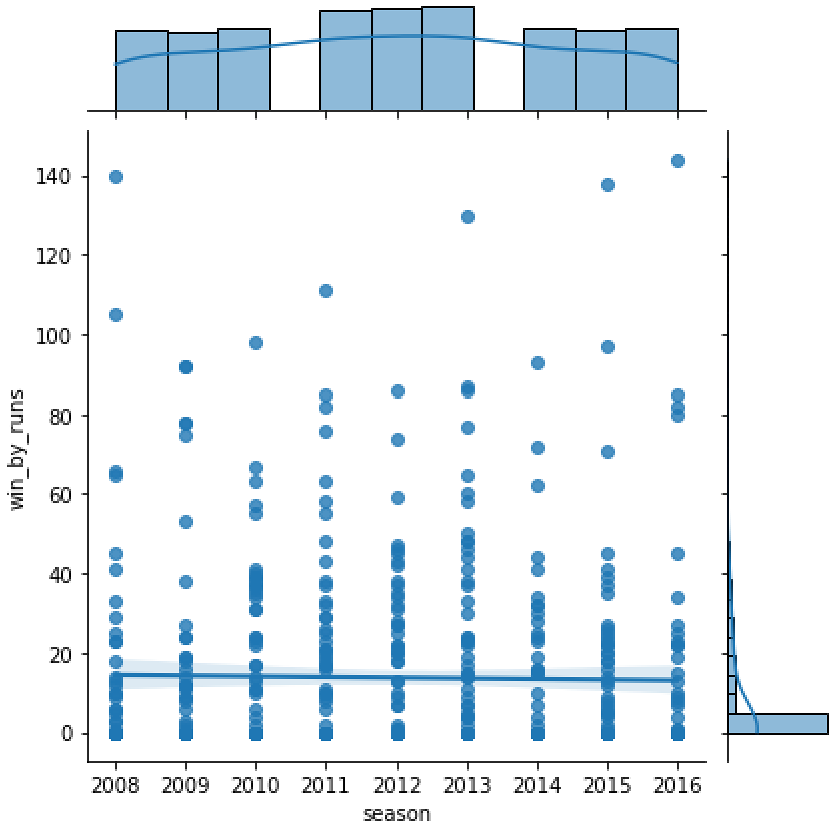
: <AxesSubplot:>



#jointplot is used to analyse the correlation between the data

sns.jointplot(x='season',y='win_by_runs',data=ipl1,kind='reg')

```
<seaborn.axisgrid.JointGrid at 0x1dd8dda1400>
```



INTERPRETATION

Presence of outliers.

4. Perform EDA on company_sales_data.csv with proper interpretation based on the visualization.

a. Read all product sales data and show it using a multiline plot.

b. Read toothpaste sales data of each month and show it using a scatter plot.

c. Read face cream and facewash product sales data and show it using the bar chart.

d. Read the total profit of each month and show it using the histogram to see the most common profit ranges.

e. Calculate total sale data for last year for each product and show it using a Pie chart.

import matplotlib.pyplot as plt

%matplotlib inline

data=pd.read_csv("C:\\python_code\\company_sales_data - company_sales_data.csv")

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   month_number  12 non-null     int64
 1   facecream     12 non-null     int64
 2   facewash      12 non-null     int64
 3   toothpaste    12 non-null     int64
 4   bathingsoap   12 non-null     int64
 5   shampoo       12 non-null     int64
 6   moisturizer   12 non-null     int64
 7   total_units   12 non-null     int64
 8   total_profit  12 non-null     int64
dtypes: int64(9)
memory usage: 992.0 bytes
```

data.columns

```
Index(['month_number', 'facecream', 'facewash', 'toothpaste', 'bathingsoap',
       'shampoo', 'moisturizer', 'total_units', 'total_profit'],
      dtype='object')
```

---

facecream=data['facecream']

facewash=data['facewash']

toothpaste=data['toothpaste']

bathingsoap=data['bathingsoap']

moisturizer=data['moisturizer']

shampoo=data['shampoo']

month_number=data['month_number']

plt.plot(month_number,facecream, linestyle='-',color='red', marker='o')

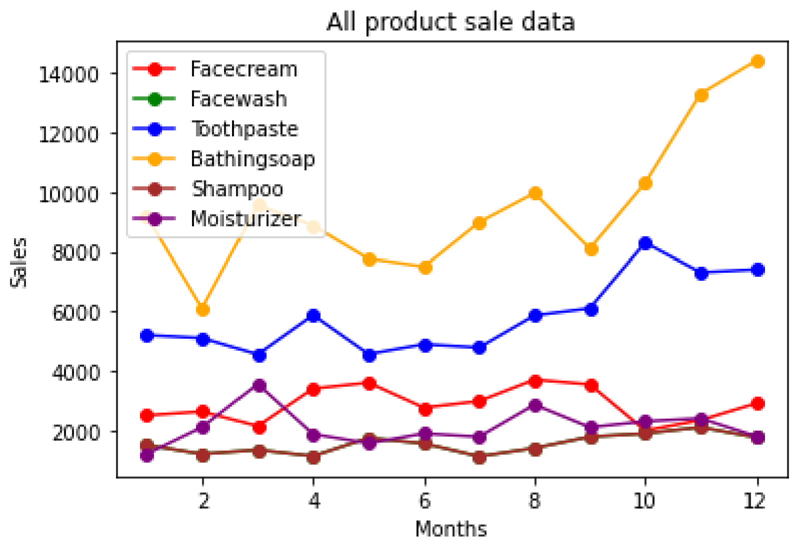
plt.plot(month_number,facewash, linestyle='-',color='green', marker='o')

plt.plot(month_number,toothpaste, linestyle='-',color='blue', marker='o')

plt.plot(month_number, bathingsoap , linestyle='-',color='orange', marker='o')

plt.plot(month_number,moisturizer, linestyle='-',color='brown',marker='o')

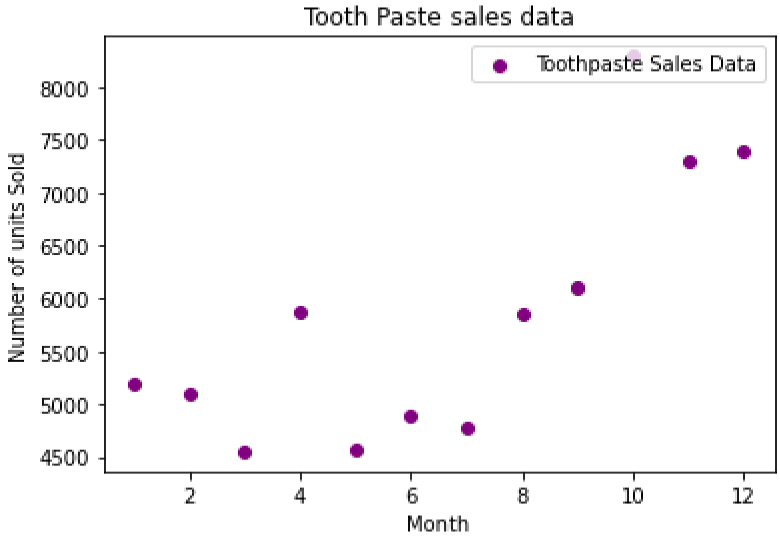
```
plt.plot(month_number,shampoo, linestyle='-', color='purple',marker='o' )

plt.title("All product sale data")

plt.xlabel("Months")

plt.ylabel("Sales")

plt.legend(labels = ['Facecream', 'Facewash', 'Toothpaste',
'Bathingsoap','Shampoo','Moisturizer'],loc="upper left")

plt.show()
```



INTERPRETATION

Sales increased for the product bathingsoap, and decreased for shampoo.

```
plt.scatter(month_number,toothpaste, color="purple")

plt.title("Tooth Paste sales data")

plt.xlabel("Month")

plt.ylabel("Number of units Sold")

plt.legend(["Toothpaste Sales Data"],loc="upper right")

plt.show()
```



INTERPRETATION

Toothpaste Sales is increased over the months.

plt.bar(data["month_number"] - 0.25, data["facecream"], width=0.25, color="blue",label="Face Cream sales data", align="edge")

plt.bar(data["month_number"] + 0.25, data["facewash"], width=-0.25, color="red", label="Face Wash sales data", align="edge")

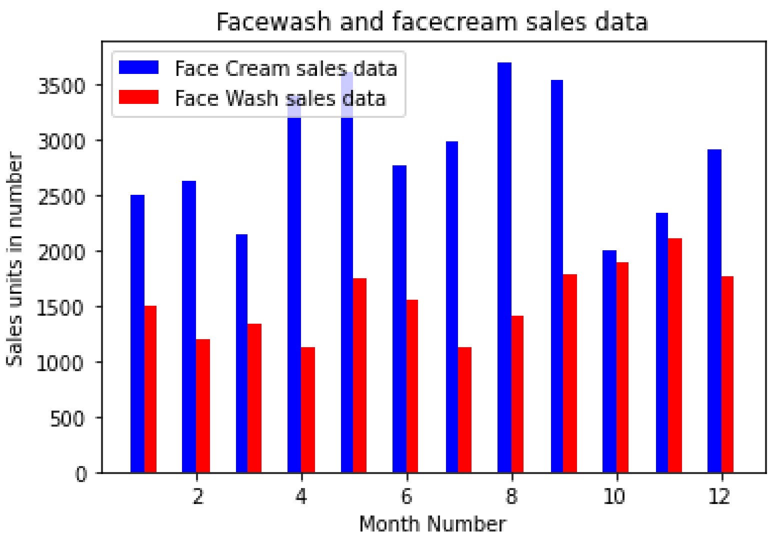
plt.title("Facewash and facecream sales data")

plt.xlabel("Month Number")

plt.ylabel("Sales units in number")

plt.legend(loc="upper left")

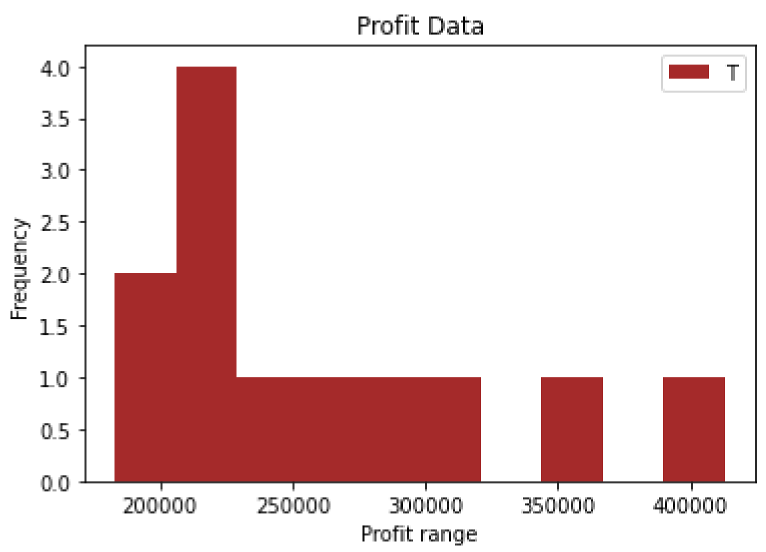
plt.show()



INTERPRETATION

Face cream sales is increased compared to face wash over the months.

```
total_profit=data['total_profit']
plt.hist(total_profit,bins=10,density=False,color="brown")
plt.title("Profit Data")
plt.xlabel("Profit range")
plt.ylabel("Frequency")
plt.legend("Total profit")
plt.show()
```



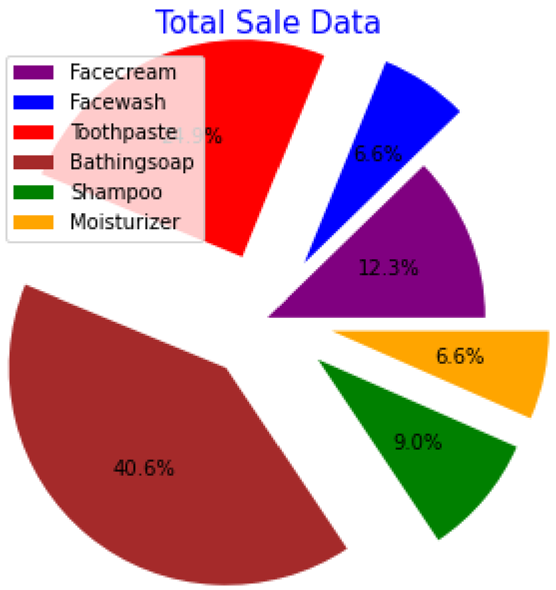
INTERPRETATION

Profit reaches around 220000 with the frequency 4.0 with higher density.

```
colors=['purple','blue','red','brown','green','orange']
explode=[0,0.3,0.3,0.3,0.3,0.3]
plt.figure(figsize=(5,5))
total_sale = [sum(facecream), sum(facewash), sum(toothpaste), sum(bathingsoap),
sum(shampoo),sum(moisturizer)]
plt.pie(total_sale ,explode=explode,colors=colors,autopct='%1.1f%%')
plt.title('Total Sale Data',color="blue",fontsize=15)
```

plt.legend(labels = ['Facecream', 'Facewash', 'Toothpaste', 'Bathingsoap','Shampoo','Moisturizer'])

plt.show()



INTERPRETATION

Sales of bathingsoap is increased compared to other product 40.6.

Facewash product is the least sales data.

5. Perform the classification on the breast cancer using four different algorithms:

A. Analyse the performance metrics of the four algorithms.

B. Interpret which algorithm gives a very good accuracy score and why?

C. Remove the target variable column and perform clustering by choosing k value using elbow method. Apply the cluster label as target and perform classification using the best model from 5A. Analyse the performance metrics of clustering.

D. Perform clustering customer dataset by using elbow method. By providing the Cluster label as target column perform classification and analyse its performance metrics.

#5A. Perform the classification on the breast cancer using four different algorithms:

from sklearn import datasets

cancer=datasets.load_breast_cancer()

df_cancer = pd.DataFrame(cancer.data, columns=cancer.feature_names) df_cancer['target'] = pd.Series(cancer.target)

df_cancer

X=df_cancer.drop('target',axis=1)

y=df_cancer['target']

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=101)

from sklearn.linear_model import LinearRegression

lm=LinearRegression()

lm.fit(X_train,y_train)

```
LinearRegression()
```

print(lm.intercept_)

```
2.8791124302214155
```

print(lm.coef_)

```
[ 1.88483067e-01 -1.10810198e-03 -1.92103818e-02 -2.87082019e-04
  1.66060120e+00  4.22854350e+00 -1.77440725e+00 -2.09765541e+00
 -7.62432719e-01  5.37073957e-01 -7.91902416e-01  1.06334753e-02
  6.84641664e-02  8.15822173e-04 -9.58224490e+00 -1.15873529e-01
  4.27311559e+00 -1.13627936e+01 -3.28027537e+00  6.51029908e+00
 -1.92646209e-01 -9.46135655e-03  2.67498659e-03  1.02486499e-03
 -1.71984338e+00 -1.53672081e-01 -4.05447900e-01 -5.18672336e-01
 -1.79236096e-01 -3.72493843e+00]
```

cdf=pd.DataFrame(lm.coef_,X_train.columns,columns=['Coeff'])

cdf

]:

	Coeff
mean radius	0.188483
mean texture	-0.001108
mean perimeter	-0.019210
mean area	-0.000287
mean smoothness	1.660601
mean compactness	4.228544
mean concavity	-1.774407
mean concave points	-2.097655
mean symmetry	-0.762433
mean fractal dimension	0.537074
radius error	-0.791902
texture error	0.010633
perimeter error	0.068464
area error	0.000816

prediction=lm.predict(X_test)

from sklearn.linear_model import LogisticRegression

model=LogisticRegression()

model.fit(X_train,y_train)

prediction=model.predict(X_test)

prediction

```
array([1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0])
```

from sklearn.metrics import classification_report,confusion_matrix

print("Classification Report")

```
Classification Report
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        42
           1       0.96      0.96      0.96        72

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

INTERPRETATION

Logistic regression shows 94% accuracy.

print(classification_report(y_test,prediction))

print("Confusion matrix")

print(confusion_matrix(y_test,prediction))

print(confusion_matrix(y_test,prediction))

```
              precision    recall  f1-score   support

           0       0.93      0.90      0.92        42
           1       0.95      0.96      0.95        72

    accuracy                           0.94       114
   macro avg       0.94      0.93      0.93       114
weighted avg       0.94      0.94      0.94       114

Confusion matrix
[[38  4]
 [ 3 69]]
```

from sklearn.metrics import accuracy_score

print('Accuracy Score: %.3f' % accuracy_score(y_test,prediction))

#Remove the target variable column

data=df_cancer.drop('target',axis=1)

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
```

```
# perform clustering by choosing k value using elbow method.
from sklearn.cluster import KMeans
Kmeans=KMeans(n_clusters=2)
Kmeans.fit(data)
def converter(target):
    if target=='Yes':
        return 1
    else:
        return 0
data['Cluster']=df_cancer['target'].apply(converter)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(data['Cluster'],Kmeans.labels_))
```

```
              precision    recall  f1-score   support

           0       1.00      0.23      0.37       569
           1       0.00      0.00      0.00         0

    accuracy                           0.23       569
   macro avg       0.50      0.12      0.19       569
weighted avg       1.00      0.23      0.37       569
```

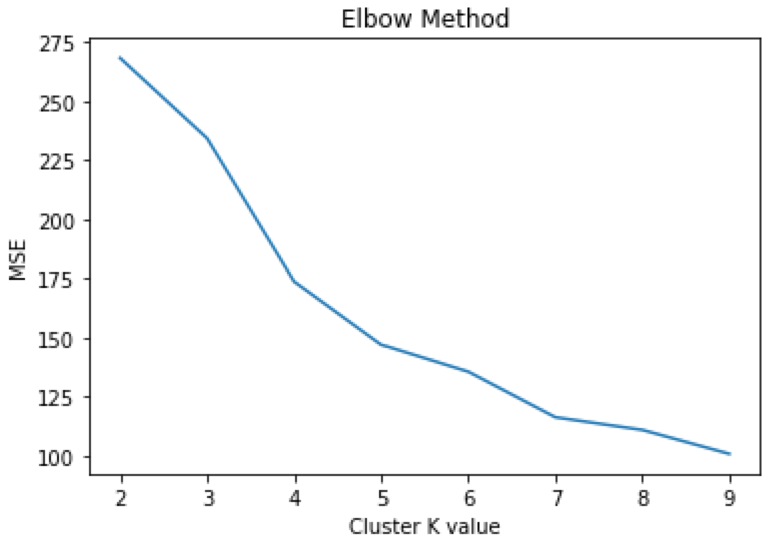
INTERPRETATION

KMeans shows 50% accuracy.

```
print(confusion_matrix(data['Cluster'],Kmeans.labels_))
```

```
[[131 438]
 [  0   0]]
```

```
#calculating mena squared error
from scipy.spatial.distance import cdist
data=data.drop('Cluster',axis=1)
distortion=[]
K=range(2,10)
for k in K:
    kmeans=KMeans(n_clusters=k)
    kmeans.fit(data)
    mse=sum(np.min(cdist(data,kmeans.cluster_centers_,'euclidean'),axis=1))/data.shape[0]
    distortion.append(mse)
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(K,distortion)
plt.xlabel('Cluster K value')
plt.ylabel('MSE')
plt.title('Elbow Method')
plt.show()
```



<u>INTERPRETATION</u>

As K value increases mean squared error dectreases.

from sklearn.neighbors import KNeighborsClassifier

knn_model=KNeighborsClassifier(n_neighbors=1) #(increase in neighbor decrease in accuracy)

knn_model.fit(X_train,Y_train)

pred=knn_model.predict(X_test)

from sklearn.metrics import classification_report,confusion_matrix

print(classification_report(Y_test,pred))

print(confusion_matrix(Y_test,pred))

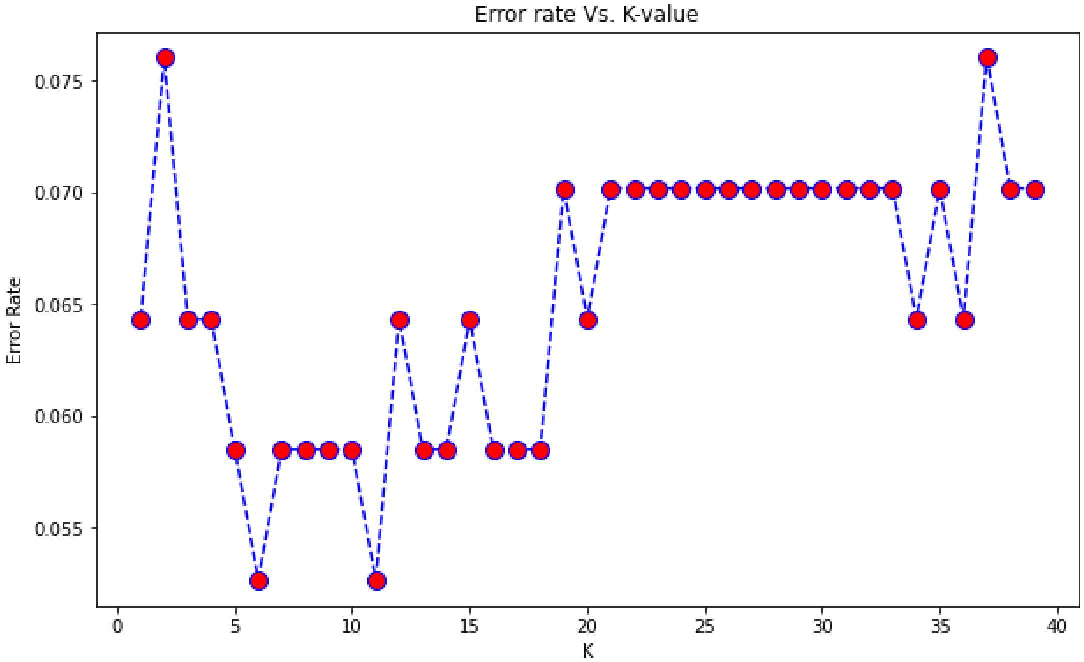
```
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        59
           1       0.95      0.95      0.95       112

    accuracy                           0.94       171
   macro avg       0.93      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171

[[ 54   5]
 [  6 106]]
```

<u>INTERPRETATION</u>

KNN shows 93% accuracy.

```
error_rate=[]

for i in range(1,40):

    knn=KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train,Y_train)

    pred_i=knn.predict(X_test)

    error_rate.append(np.mean(pred_i!=Y_test))

plt.figure(figsize=(10,6))

plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='o',markerfacecolor='red',
markersize=10)

plt.title('Error rate Vs. K-value')

plt.xlabel('K')

plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



INTERPRETATION

Error rate is minimal between the range of 20 and 35 with higher accuracy rate.

```
from sklearn.tree import DecisionTreeClassifier

clf=DecisionTreeClassifier()

clf=clf.fit(X_train,Y_train)
```

pred_tree=clf.predict(X_test)

print(classification_report(Y_test,pred_tree))

print(confusion_matrix(Y_test,pred_tree))

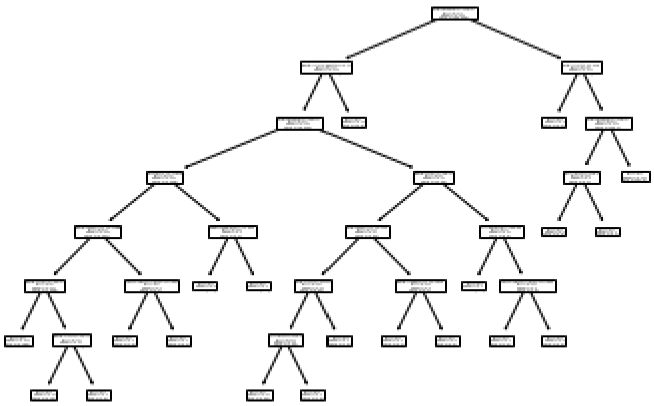
```
              precision    recall  f1-score   support

           0       0.83      0.93      0.88        59
           1       0.96      0.90      0.93       112

    accuracy                           0.91       171
   macro avg       0.90      0.92      0.91       171
weighted avg       0.92      0.91      0.91       171

[[ 55   4]
 [ 11 101]]
```

INTERPRETATION

Decision tree shows 90% accuracy.

from sklearn import tree

tree.plot_tree(clf,feature_names=X_train.columns)



B.**Logistic regression is the best model.**

D.

import pandas as pd

import numpy as np

cu=pd.read_csv("C:\\python_code\\segmented_customers - segmented_customers.csv")

cu

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	cluster
0	1	Male	19	15	39	4
1	2	Male	21	15	81	3
2	3	Female	20	16	6	4
3	4	Female	23	16	77	3
4	5	Female	31	17	40	4
...	...	...	...	...	...	...
195	196	Female	35	120	79	1
196	197	Female	45	126	28	2
197	198	Male	32	126	74	1
198	199	Male	32	137	18	2
199	200	Male	30	137	83	1

200 rows × 6 columns

cu.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
 5   cluster                 200 non-null    int64
dtypes: int64(5), object(1)
memory usage: 9.5+ KB
```

#to fetch the columns with numerical data types

cu_numeric=cu.select_dtypes(exclude=['object'])

cu_numeric

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)	cluster
0	1	19	15	39	4
1	2	21	15	81	3
2	3	20	16	6	4
3	4	23	16	77	3
4	5	31	17	40	4
...	...	...	...	...	...
195	196	35	120	79	1
196	197	45	126	28	2
197	198	32	126	74	1
198	199	32	137	18	2
199	200	30	137	83	1

200 rows × 5 columns

to find number of observations,number of columns and missing value if any

cu.columns

```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)', 'cluster'],
      dtype='object')
```

#to remove whitespace

cu.columns=cu.columns.str.strip()

cu.columns

```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)', 'cluster'],
      dtype='object')
```

#to replace whitespace with underscore

cu.columns=cu.columns.str.replace(' ','_')

cu.columns

```
Index(['CustomerID', 'Gender', 'Age', 'Annual_Income_(k$)',
       'Spending_Score_(1-100)', 'cluster'],
      dtype='object')
```

cu_numeric.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Age                     200 non-null    int64
 2   Annual Income (k$)      200 non-null    int64
 3   Spending Score (1-100)  200 non-null    int64
 4   cluster                 200 non-null    int64
dtypes: int64(5)
memory usage: 7.9 KB
```

cu.isnull().sum()

```
CustomerID                0
Gender                    0
Age                       0
Annual_Income_(k$)        0
Spending_Score_(1-100)    0
cluster                   0
dtype: int64
```

#removing the class label(age)

data=cu.drop('Age',axis=1)

data=cu.drop('Gender',axis=1)

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Age                     200 non-null    int64
 2   Annual_Income_(k$)      200 non-null    int64
 3   Spending_Score_(1-100)  200 non-null    int64
 4   cluster                 200 non-null    int64
dtypes: int64(5)
memory usage: 7.9 KB
```

from sklearn.cluster import KMeans

kmeans=KMeans(n_clusters=2)# creating 2 cluster

kmeans.fit(data)

```
KMeans(n_clusters=2)
```

#to find center of the data

kmeans.cluster_centers_

```
array([[150.        ,  37.77227723,  81.35643564,  50.45544554,
          1.82178218],
       [ 50.        ,  39.94949495,  39.34343434,  49.93939394,
          2.67676768]])
```

kmeans.labels_

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0])
```

from sklearn.metrics import classification_report,confusion_matrix

print(classification_report(data['cluster'],kmeans.labels_))

	precision	recall	f1-score	support
0	0.33	0.73	0.46	45
1	0.39	1.00	0.56	39
2	0.00	0.00	0.00	35
3	0.00	0.00	0.00	22
4	0.00	0.00	0.00	21
5	0.00	0.00	0.00	38
accuracy			0.36	200
macro avg	0.12	0.29	0.17	200
weighted avg	0.15	0.36	0.21	200

INTERPRETATION

KMeans shows 12% accuracy.

print(confusion_matrix(data['cluster'],kmeans.labels_))

```
[[33 12  0  0  0  0]
 [ 0 39  0  0  0  0]
 [ 0 35  0  0  0  0]
 [22  0  0  0  0  0]
 [21  0  0  0  0  0]
 [23 15  0  0  0  0]]
```

```
#calculating mean squared error(mse)
from scipy.spatial.distance import cdist
data.shape
```

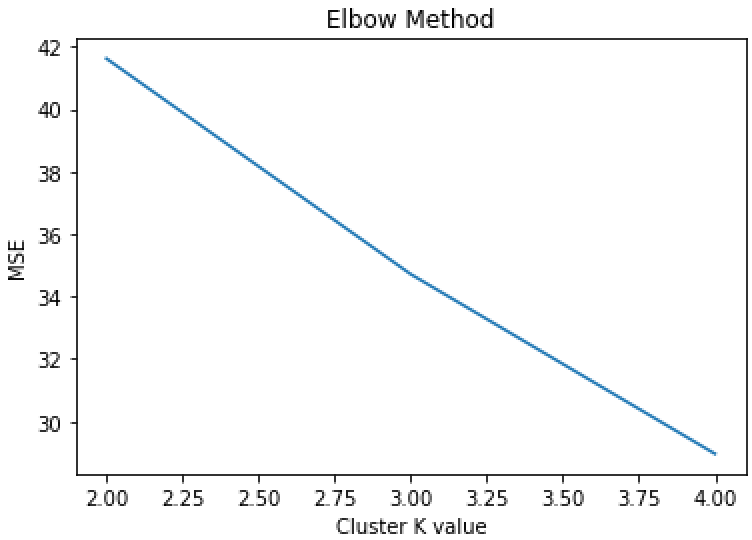
```
(200, 5)
```

```
data.shape[0]
```

```
200
```

```
distortion=[]
K=range(2,5)
for k in K:
    kmeans=KMeans(n_clusters=k)
    kmeans.fit(data)
    mse=sum(np.min(cdist(data,kmeans.cluster_centers_,'euclidean'),axis=1))/data.shape[0]
    distortion.append(mse)
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(K,distortion)
plt.xlabel('Cluster K value')
plt.ylabel('MSE')
plt.title('Elbow Method')
plt.show#error decreases as K value increases
```

<function matplotlib.pyplot.show(close=None, block=None)>



<u>INTERPRETATION</u>

As K value increases error rate decreases with high precision rate.

```
target=[]
for i in range(len(data['cluster'])):
    if data['cluster'][i]==1:
        target.append('A')
    elif data['cluster'][i]==2:
        target.append('B')
    elif data['cluster'][i]==3:
        target.append('C')
    else:
        target.append('D')

data['target']=target
data
```

	CustomerID	Age	Annual_Income_(k$)	Spending_Score_(1-100)	cluster	target
0	1	19	15	39	4	D
1	2	21	15	81	3	C
2	3	20	16	6	4	D
3	4	23	16	77	3	C
4	5	31	17	40	4	D
...	...	...	...	...	...	...
195	196	35	120	79	1	A
196	197	45	126	28	2	B
197	198	32	126	74	1	A
198	199	32	137	18	2	B
199	200	30	137	83	1	A

200 rows × 6 columns

x=data.drop(['cluster','target'],axis=1)

y=data['cluster']

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=101)#80%-train,20%-test,101=seed point

#create the model and train the model

from sklearn.linear_model import LinearRegression

lm=LinearRegression()

lm.fit(x_train,y_train)

```
LinearRegression()
```

#evaluate the model

print(lm.intercept_)

```
8.350586398901937
```

print(lm.coef_)#coeff of 5 independent variables in the dataset

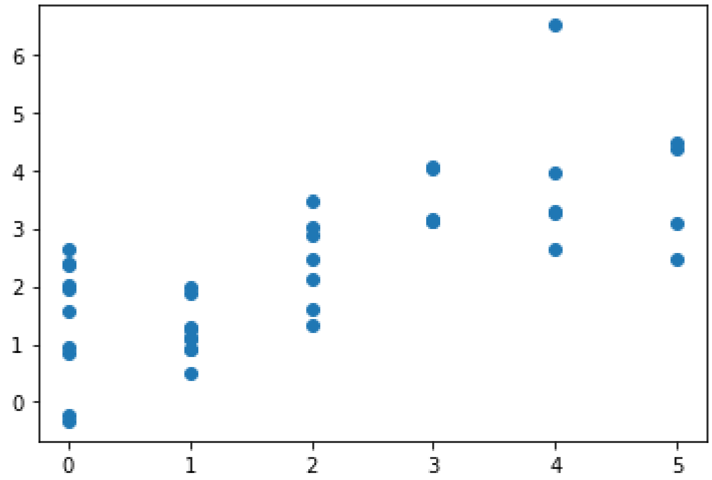
```
[-0.01725505 -0.09080864  0.01264395 -0.03049777]
```

pred=lm.predict(x_test)

testing_prediction=lm.predict(x_test)

plt.scatter(y_test,testing_prediction)#more or less similar

```
<matplotlib.collections.PathCollection at 0x1ff6b0f7490>
```



from sklearn import metrics

metrics.mean_absolute_error(y_test,testing_prediction)

```
0.9278475527645952
```

metrics.mean_squared_error(y_test,testing_prediction)

```
1.4943365115159322
```

import numpy as np

rmse=np.sqrt(metrics.mean_squared_error(y_test,testing_prediction))

print(rmse)

```
1.222430575335848
```

6. A. Consider you are provided with the planet's dataset from seaborn library. The dataset gives information on planets that astronomers have discovered around other stars (known as extrasolar planets or exoplanets for short). Perform the following operations on the dataset

Load the dataset and display top 10 records and bottom 5 records

o Statistically analyse the overall properties of the dataset using a single command after

o dropping null or missing values

o Find the mean value of orbital periods (in days) that each method is sensitive to.

o Perform multiple aggregation like min, max and mean on the column orbital period.

o Remove the column method from the dataset

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import scipy

import seaborn as sns

print(sns.get_dataset_names())

```
['anagrams', 'anscombe', 'attention', 'brain_networks', 'car_crashes', 'diamonds', 'dots', 'dowjones', 'exercise', 'flights',
'fmri', 'geyser', 'glue', 'healthexp', 'iris', 'mpg', 'penguins', 'planets', 'seaice', 'taxis', 'tips', 'titanic']
```

#Load the dataset and display top 10 records and bottom 5 records

df=sns.load_dataset('planets')

df.head(10)

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009
5	Radial Velocity	1	185.840	4.80	76.39	2008
6	Radial Velocity	1	1773.400	4.64	18.15	2002
7	Radial Velocity	1	798.500	NaN	21.41	1996
8	Radial Velocity	1	993.300	10.30	73.10	2008
9	Radial Velocity	2	452.800	1.99	74.79	2010

df.tail(5)

	method	number	orbital_period	mass	distance	year
1030	Transit	1	3.941507	NaN	172.0	2006
1031	Transit	1	2.615864	NaN	148.0	2007
1032	Transit	1	3.191524	NaN	174.0	2007
1033	Transit	1	4.125083	NaN	293.0	2008
1034	Transit	1	4.187757	NaN	260.0	2008

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1035 entries, 0 to 1034
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   method          1035 non-null   object 
 1   number          1035 non-null   int64  
 2   orbital_period  992 non-null    float64
 3   mass            513 non-null    float64
 4   distance        808 non-null    float64
 5   year            1035 non-null   int64  
dtypes: float64(3), int64(2), object(1)
memory usage: 48.6+ KB
```

#dropping null or missing values

df.isnull().sum()

```
method              0
number              0
orbital_period     43
mass              522
distance          227
year                0
dtype: int64
```

df.dropna()

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.30000	7.100	77.40	2006
1	Radial Velocity	1	874.77400	2.210	56.95	2008
2	Radial Velocity	1	763.00000	2.600	19.84	2011
3	Radial Velocity	1	326.03000	19.400	110.62	2007
4	Radial Velocity	1	516.22000	10.500	119.47	2009
...	...	...	...	...	...	...
640	Radial Velocity	1	111.70000	2.100	14.90	2009
641	Radial Velocity	1	5.05050	1.068	44.46	2013
642	Radial Velocity	1	311.28800	1.940	17.24	1999
649	Transit	1	2.70339	1.470	178.00	2013
784	Radial Velocity	3	580.00000	0.947	135.00	2012

498 rows × 6 columns

#Statistically analyse the overall properties of the dataset using a single command after

df.describe()

	number	orbital_period	mass	distance	year
count	1035.000000	992.000000	513.000000	808.000000	1035.000000
mean	1.785507	2002.917596	2.638161	264.069282	2009.070531
std	1.240976	26014.728304	3.818617	733.116493	3.972567
min	1.000000	0.090706	0.003600	1.350000	1989.000000
25%	1.000000	5.442540	0.229000	32.560000	2007.000000
50%	1.000000	39.979500	1.260000	55.250000	2010.000000
75%	2.000000	526.005000	3.040000	178.500000	2012.000000
max	7.000000	730000.000000	25.000000	8500.000000	2014.000000

#Find the mean value of orbital periods (in days) that each method is sensitive to.

df.groupby('method')['orbital_period'].mean()

```
method
Astrometry                            631.180000
Eclipse Timing Variations            4751.644444
Imaging                            118247.737500
Microlensing                         3153.571429
Orbital Brightness Modulation           0.709307
Pulsar Timing                        7343.021201
Pulsation Timing Variations          1170.000000
Radial Velocity                       823.354680
Transit                                21.102073
Transit Timing Variations              79.783500
Name: orbital_period, dtype: float64
```

#Perform multiple aggregation like min, max and mean on the column orbital period.

df.agg({'orbital_period':{'mean','min','max'}})

	orbital_period
max	730000.000000
min	0.090706
mean	2002.917596

#Remove the column method from the dataset

del df['mass']

df

	method	number	orbital_period	distance	year
0	Radial Velocity	1	269.300000	77.40	2006
1	Radial Velocity	1	874.774000	56.95	2008
2	Radial Velocity	1	763.000000	19.84	2011
3	Radial Velocity	1	326.030000	110.62	2007
4	Radial Velocity	1	516.220000	119.47	2009
...	...	...	...	...	...
1030	Transit	1	3.941507	172.00	2006
1031	Transit	1	2.615864	148.00	2007
1032	Transit	1	3.191524	174.00	2007
1033	Transit	1	4.125083	293.00	2008
1034	Transit	1	4.187757	260.00	2008

1035 rows × 5 columns

B. Perform the Quantitative analysis on the Brazillian fire dataset which has the following information like:

CO2

o Load the dataset and display top 10 records and bottom 5 records

o Statistically analyse the overall properties of the dataset using a single command after

o dropping null or missing values

o Find the mean value of orbital periods (in days) that each method is sensitive to.

o Perform multiple aggregation like min, max and mean on the column orbital period.

o Remove the column method from the dataset

B. Perform the Quantitative analysis on the Brazillian fire dataset which has the following information like:

Year: when the fire occurred

State: where the fire was reported

Month: when the fire occurred

Number of fires: frequency reported

Date reported: when the fire was reported.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy
import seaborn as sns
df=sns.load_dataset('planets')
print(df.head(5))
```

```
           method  number  orbital_period   mass  distance  year
0  Radial Velocity       1         269.300   7.10     77.40  2006
1  Radial Velocity       1         874.774   2.21     56.95  2008
2  Radial Velocity       1         763.000   2.60     19.84  2011
3  Radial Velocity       1         326.030  19.40    110.62  2007
4  Radial Velocity       1         516.220  10.50    119.47  2009
```

```
import matplotlib.pyplot as plt
import seaborn as sns
data=pd.read_csv("E:\\python code\Brazilian-fire-dataset.csv")
data
```

	Year	State	Month	Number of Fires	Date Reported
0	1998	Acre	January	0.0	1/01/1998
1	1999	Acre	January	0.0	1/01/1999
2	2000	Acre	January	0.0	1/01/2000
3	2001	Acre	January	0.0	1/01/2001
4	2002	Acre	January	0.0	1/01/2002
...	...	...	...	...	...
6449	2012	Tocantins	December	128.0	1/01/2012
6450	2013	Tocantins	December	85.0	1/01/2013
6451	2014	Tocantins	December	223.0	1/01/2014
6452	2015	Tocantins	December	373.0	1/01/2015
6453	2016	Tocantins	December	119.0	1/01/2016

6454 rows × 5 columns

```
data.isnull().sum()
```

```
Year               0
State              0
Month              0
Number of Fires    0
Date Reported      0
dtype: int64
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6454 entries, 0 to 6453
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Year             6454 non-null   int64
 1   State            6454 non-null   object
 2   Month            6454 non-null   object
 3   Number of Fires  6454 non-null   float64
 4   Date Reported    6454 non-null   object
dtypes: float64(1), int64(1), object(3)
memory usage: 252.2+ KB
```

data.columns

```
Index(['Year', 'State', 'Month', 'Number of Fires', 'Date Reported'], dtype='object')
```

data.columns=data.columns.str.strip()

data.columns

```
Index(['Year', 'State', 'Month', 'Number of Fires', 'Date Reported'], dtype='object')
```

data.columns=data.columns.str.replace(' ','_')

data.columns

```
Index(['Year', 'State', 'Month', 'Number_of_Fires', 'Date_Reported'], dtype='object')
```

data.info()

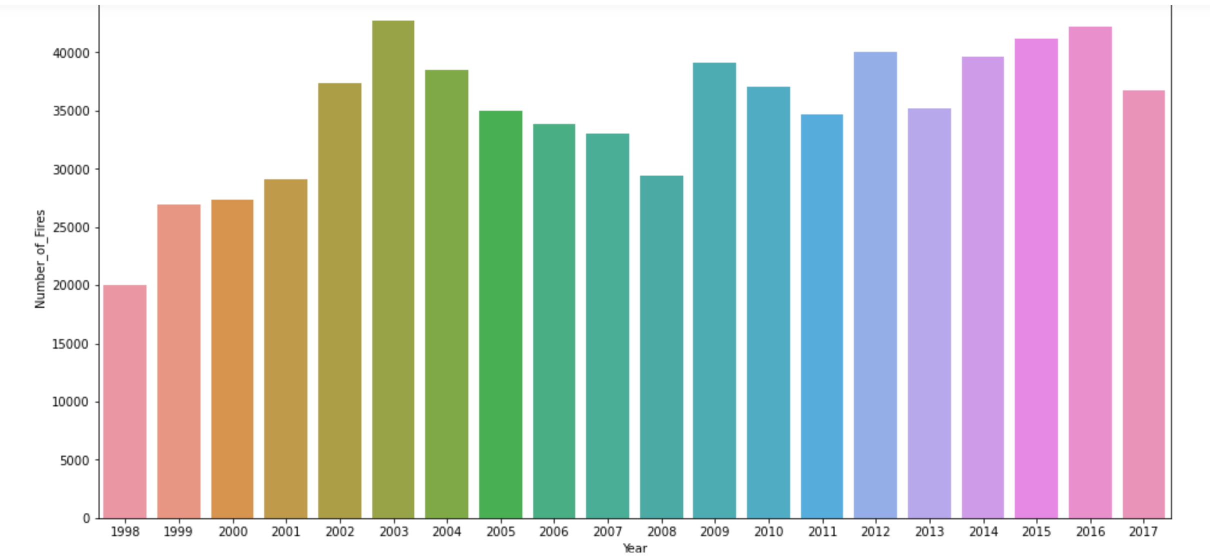
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6454 entries, 0 to 6453
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Year             6454 non-null   int64
 1   State            6454 non-null   object
 2   Month            6454 non-null   object
 3   Number_of_Fires  6454 non-null   float64
 4   Date_Reported    6454 non-null   object
dtypes: float64(1), int64(1), object(3)
memory usage: 252.2+ KB
```

year=data.groupby('Year')['Number_of_Fires'].sum().reset_index()

print(year)

	Year	Number_of_Fires
0	1998	20013.971
1	1999	26882.821
2	2000	27351.251
3	2001	29071.612
4	2002	37390.600
5	2003	42760.674
6	2004	38453.163
7	2005	35004.965
8	2006	33832.161
9	2007	33037.413
10	2008	29378.964
11	2009	39117.178
12	2010	37037.449
13	2011	34633.545
14	2012	40084.860
15	2013	35146.118
16	2014	39621.183
17	2015	41208.292
18	2016	42212.229
19	2017	36685.624

```
plt.figure(figsize = (16,8))
sns.barplot(x="Year", y="Number_of_Fires", data=year)
plt.show()
```



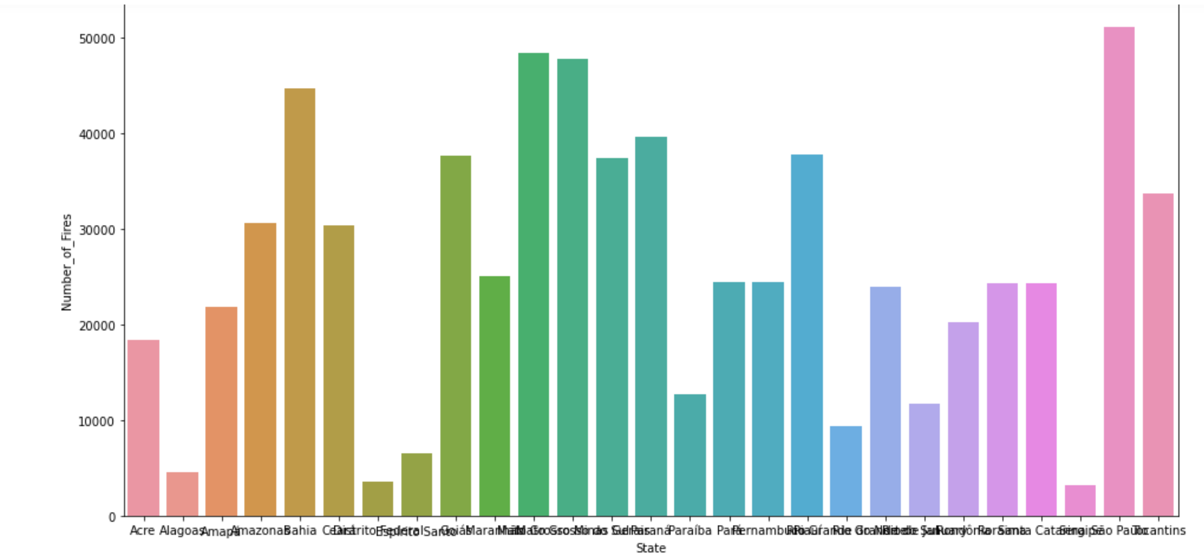
INTERPRETATION

Number of fires is increased in 2003,2016.

```
state=data.groupby('State')['Number_of_Fires'].sum().reset_index()
print(state)
```

	State	Number_of_Fires
0	Acre	18464.030
1	Alagoas	4644.000
2	Amapá	21831.576
3	Amazonas	30650.129
4	Bahia	44746.226
5	Ceará	30428.063
6	Distrito Federal	3561.000
7	Espírito Santo	6546.000
8	Goiás	37695.520
9	Maranhão	25129.131
10	Mato Grosso	48477.827
11	Mato Grosso do Sul	47768.201
12	Minas Gerais	37475.258
13	Paraná	39648.918
14	Paraíba	12787.000
15	Pará	24512.144
16	Pernambuco	24498.000
17	Piauí	37803.747
18	Rio Grande do Norte	9426.000
19	Rio Grande do Sul	24031.865
20	Rio de January	11703.000
21	Rondônia	20285.429
22	Roraima	24385.074
23	Santa Catarina	24359.852
24	Sergipe	3237.000

```
plt.figure(figsize = (16,8))
sns.barplot(x="State", y="Number_of_Fires", data=state)
plt.show()
```



INTERPRETATION

Number of fires is increased in different state.

month=data.groupby('Month')['Number_of_Fires'].sum().reset_index()

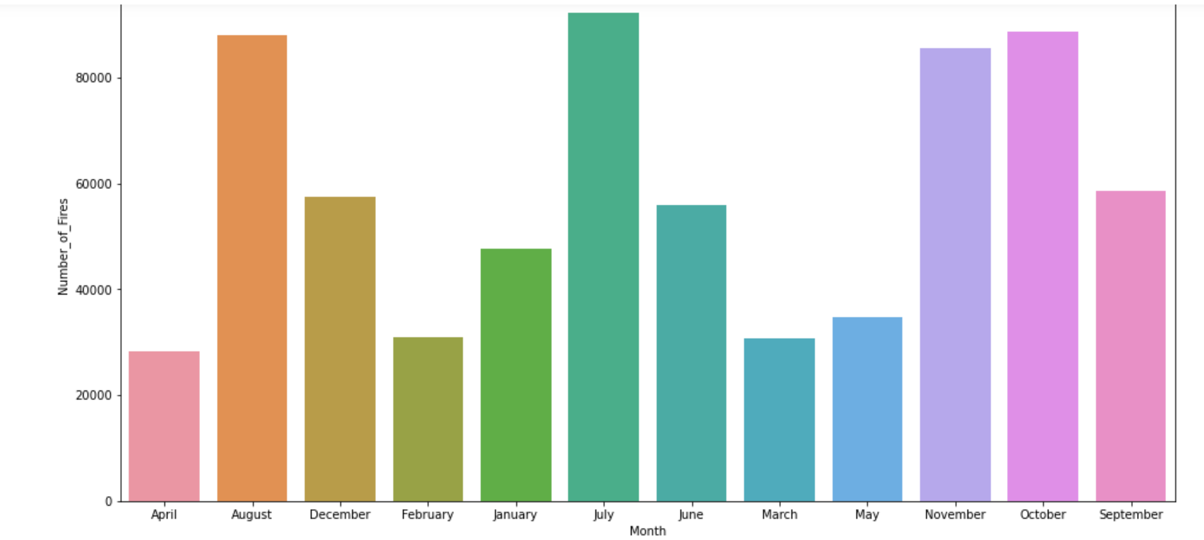
print(month)

```
       Month  Number_of_Fires
0      April        28188.770
1     August        88050.435
2   December        57535.480
3   February        30848.050
4    January        47747.844
5       July        92326.113
6       June        56010.675
7      March        30717.405
8        May        34731.363
9   November        85508.054
10   October        88681.579
11 September        58578.305
```

plt.figure(figsize = (16,8))

sns.barplot(x="Month", y="Number_of_Fires", data=month)

plt.show()



INTERPRETATION

Number of fire is increased in the month of July a d declines in the month of march

data['Number_of_Fires'].describe()

```
count    6454.000000
mean      108.293163
std       190.812242
min         0.000000
25%         3.000000
50%        24.000000
75%       113.000000
max       998.000000
Name: Number_of_Fires, dtype: float64
```

---

date=data.groupby('Date_Reported')['Number_of_Fires'].sum().reset_index()

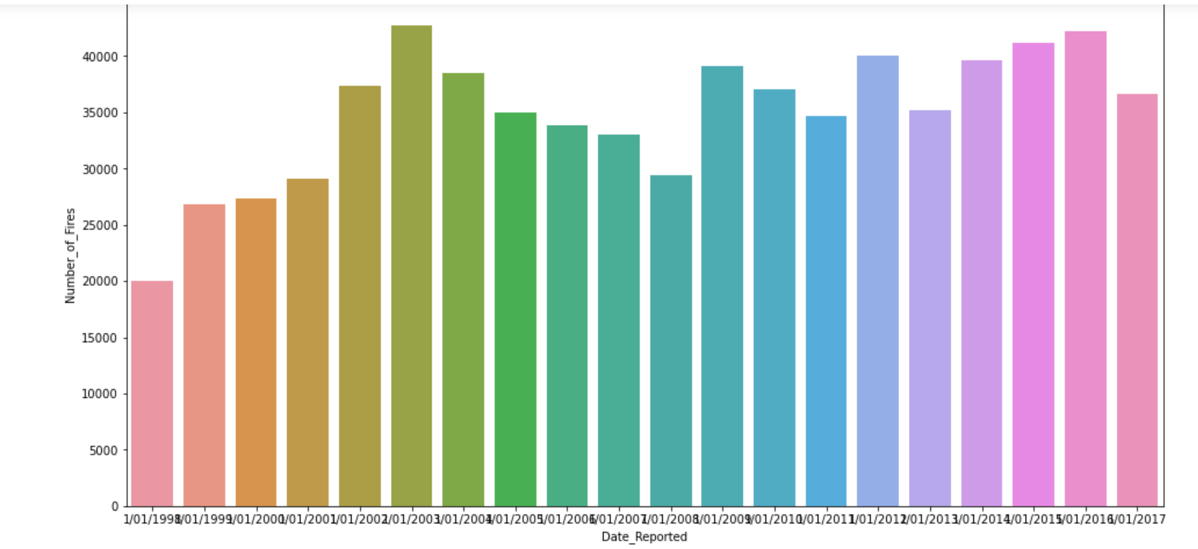
print(date)

```
    Date_Reported  Number_of_Fires
0       1/01/1998        20013.971
1       1/01/1999        26882.821
2       1/01/2000        27351.251
3       1/01/2001        29071.612
4       1/01/2002        37390.600
5       1/01/2003        42760.674
6       1/01/2004        38453.163
7       1/01/2005        35004.965
8       1/01/2006        33832.161
9       1/01/2007        33037.413
10      1/01/2008        29378.964
11      1/01/2009        39117.178
12      1/01/2010        37037.449
13      1/01/2011        34633.545
14      1/01/2012        40084.860
15      1/01/2013        35146.118
16      1/01/2014        39621.183
17      1/01/2015        41208.292
18      1/01/2016        42212.229
19      1/01/2017        36685.624
```

plt.figure(figsize = (16,8))

sns.barplot(x="Date_Reported", y="Number_of_Fires", data=date)

plt.show()



7. Perform the following TensorFlow operations on matrix and a constant

i) Addition

ii) Subtraction

iii) Multiplication

iv) Division

```
import tensorflow as tf
with tf.compat.v1.Session() as sess:

#ADDITION
    a=tf.constant (15)
    b=tf.constant (3)
    c=tf.add(a,b)
    d=sess.run(c)
    print(d)
#SUBTRACTION
    e=tf.subtract(a,b)
    f=sess.run(e)
    print(f)
```

```
#MULTIPLICATION
  g=tf.multiply(a,b)
  h=sess.run(g)
  print(h)
#DIVISION
  i=tf.divide(a,b)
  j=sess.run(i)
  print(j)
```

```
18
12
45
5.0
```

8. Perform AutoML model on the following datasets

A. advertisement dataset.

B. house price prediction dataset.

C. Titanic dataset.

Plot the Leader board of 10 best algorithms with accuracy score. Interpret your results.

A.

```
pip install auto-sklearn
import pandas as pd
import numpy as numpy
import autosklearn.regression
df=pd.read_csv("/content/Advertising - Advertising.csv")
df
```

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9
...	...	...	...	...	...
195	196	38.2	3.7	13.8	7.6
196	197	94.2	4.9	8.1	9.7
197	198	177.0	9.3	6.4	12.8
198	199	283.6	42.0	66.2	25.5

✓ 3s completed at 3:34 PM

```
df.isnull().sum()
```

```
Unnamed: 0    0
TV            0
Radio         0
Newspaper     0
Sales         0
dtype: int64
```

```
x=df.drop(["Sales"],axis=1)
y=df["Sales"]
automl=autosklearn.regression.AutoSklearnRegressor(time_left_for_this_task=5*60,per_run_time_limit=30)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=40)
automl.fit(x_train,y_train)
```

```
AutoSklearnRegressor(ensemble_class=<class 'autosklearn.ensembles.ensemble_selection.EnsembleSelection'>,
                     per_run_time_limit=30, time_left_for_this_task=300)
```

```
from sklearn.metrics import mean_absolute_error

y_pred=automl.predict(x_test)
print(automl.sprint_statistics())
```

```
auto-sklearn results:
  Dataset name: 227ab608-a3a9-11ed-8067-0242ac1c000c
  Metric: r2
  Best validation score: 0.996496
  Number of target algorithm runs: 99
  Number of successful target algorithm runs: 99
  Number of crashed target algorithm runs: 0
  Number of target algorithms that exceeded the time limit: 0
  Number of target algorithms that exceeded the memory limit: 0
```

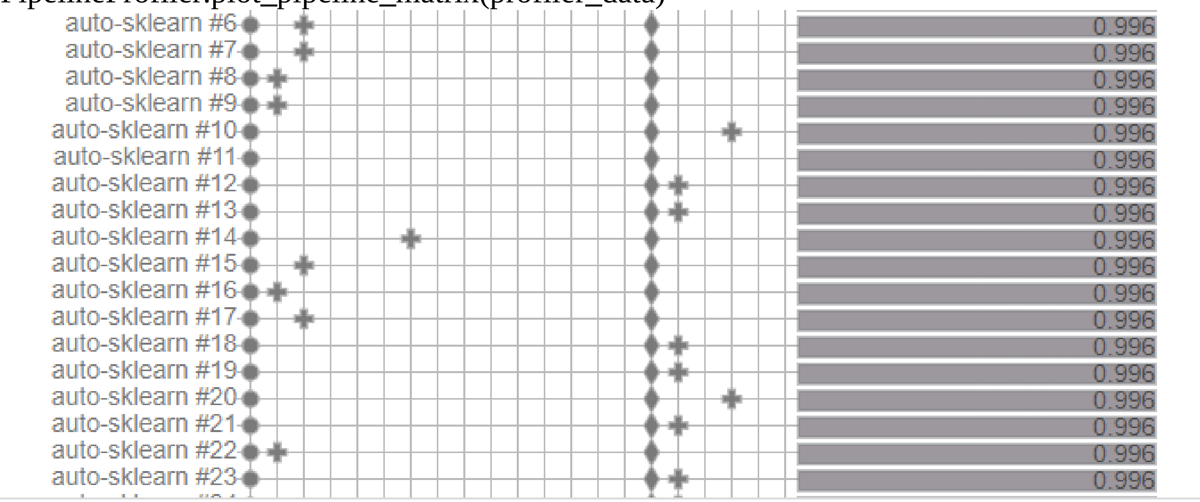
```
mae=mean_absolute_error(y_test,y_pred)
print(mae)
```

```
0.36154500253498567
```

```
print(automl.leaderboard())
```

```
          rank  ensemble_weight             type      cost  duration
model_id
66           1             0.06  gaussian_process  0.003504  0.645696
99           2             0.42  gaussian_process  0.003513  0.578452
56           3             0.02  gaussian_process  0.003581  0.637213
46           4             0.02  gaussian_process  0.003828  0.622897
86           5             0.14  gaussian_process  0.004125  0.686262
17           6             0.20  gaussian_process  0.004400  1.047740
26           7             0.02        libsvm_svr  0.005810  0.601968
24           8             0.10       extra_trees  0.012480  0.937476
31           9             0.02  gaussian_process  0.038229  0.622404
```

```
import PipelineProfiler

profiler_data=PipelineProfiler.import_autosklearn(automl)
PipelineProfiler.plot_pipeline_matrix(profiler_data)
```



3s completed at 3:34 PM

INTERPRETATION

Leaderboard shows 99% accuracy.

B.

```
pip install auto-sklearn
import pandas as pd
import numpy as numpy
import autosklearn.classification
df=pd.read_csv("/content/Housing - Housing.csv")
df
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking
0	13300000	7420	4	2	3	yes	no	no	no	yes	2
1	12250000	8960	4	4	4	yes	no	no	no	yes	3
2	12250000	9960	3	2	2	yes	no	yes	no	no	2
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2
...	...	...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	yes	no	yes	no	no	2
541	1767150	2400	3	1	1	no	no	no	no	no	0
542	1750000	3620	2	1	1	yes	no	no	no	no	0
543	1750000	2910	3	1	1	no	no	no	no	no	0

```
#to fetch the columns with numerical data types
df_numeric=df.select_dtypes(exclude=['object'])
df_numeric
```

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2
...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2
541	1767150	2400	3	1	1	0

```
df.isnull().sum()
```

```
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
dtype: int64
```

```
data=df.drop(["mainroad","guestroom","basement","hotwaterheating","airconditioning","prefarea"],axis=1)
data
```

	price	area	bedrooms	bathrooms	stories	parking	furnishingstatus
0	13300000	7420	4	2	3	2	furnished
1	12250000	8960	4	4	4	3	furnished
2	12250000	9960	3	2	2	2	semi-furnished
3	12215000	7500	4	2	2	3	furnished
4	11410000	7420	4	1	2	2	furnished
...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2	unfurnished

0s completed at 3:24 PM

```
x=data.drop(["furnishingstatus"],axis=1)
y=data["furnishingstatus"]
automl=autosklearn.classification.AutoSklearnClassifier(time_left_for_this_task=5*60,per_run_time_limit=30)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=40)
automl.fit(x_train,y_train)
```

```
AutoSklearnClassifier(ensemble_class=<class 'autosklearn.ensembles.ensemble_selection.EnsembleSelection'>,
                      per_run_time_limit=30, time_left_for_this_task=300)
```

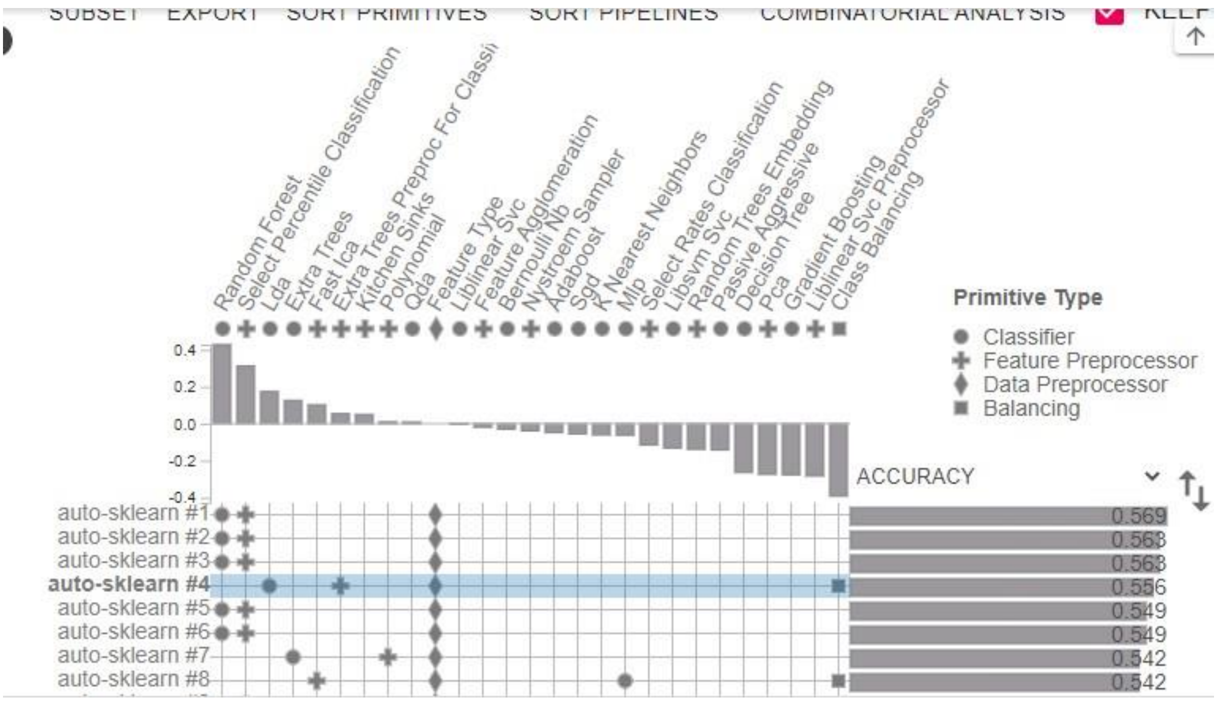
```
from sklearn.metrics import mean_absolute_error
y_pred=automl.predict(x_test)
print(automl.sprint_statistics())
```

```
auto-sklearn results:
  Dataset name: 9c5c4c85-a3a7-11ed-8204-0242ac1c000c
  Metric: accuracy
  Best validation score: 0.569444
  Number of target algorithm runs: 75
  Number of successful target algorithm runs: 75
  Number of crashed target algorithm runs: 0
  Number of target algorithms that exceeded the time limit: 0
  Number of target algorithms that exceeded the memory limit: 0
```

```
print(automl.leaderboard())
```

model_id	rank	ensemble_weight	type	cost	duration
51	1	0.24	random_forest	0.430556	1.490602
71	2	0.30	lda	0.444444	1.075507
24	3	0.04	passive_aggressive	0.472222	0.981555
44	4	0.02	libsvm_svc	0.479167	0.716349
45	5	0.04	extra_trees	0.493056	1.577940
65	6	0.02	random_forest	0.493056	1.700721
12	7	0.02	random_forest	0.500000	1.709357
27	8	0.02	mlp	0.534722	1.618086
42	9	0.12	adaboost	0.534722	1.342932
63	10	0.02	random_forest	0.562500	1.578375
34	11	0.12	adaboost	0.569444	1.101399
38	12	0.04	extra_trees	0.569444	1.652863

```
pip install PipelineProfiler
import PipelineProfiler
profiler_data=PipelineProfiler.import_autosklearn(automl)
PipelineProfiler.plot_pipeline_matrix(profiler_data)
```



INTERPRETATION

Leaderboard shows 56% accuracy.

C.

```
pip install auto-sklearn
import pandas as pd
import numpy as numpy
import autosklearn.classification
df=pd.read_csv("/content/titanic - titanic.csv")
```

```
[5] df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S

```
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
data=df.select_dtypes(exclude=['object'])
data

from sklearn.impute import SimpleImputer
imputer=SimpleImputer(strategy='mean')
data.iloc[:,:]=imputer.fit_transform(data)

data.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Age            0
SibSp          0
Parch          0
Fare           0
dtype: int64
```

```
x=data.drop(["Survived"],axis=1)
y=data["Survived"]

automl=autosklearn.classification.AutoSklearnClassifier(time_left_for_this_task=5*60,per_r
un_time_limit=30)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=40)

automl.fit(x_train,y_train)
```

```
AutoSklearnClassifier(ensemble_class=<class 'autosklearn.ensembles.ensemble_selection.EnsembleSelection'>,
                      per_run_time_limit=30, time_left_for_this_task=300)
```

```
from sklearn.metrics import mean_absolute_error
y_pred=automl.predict(x_test)
```

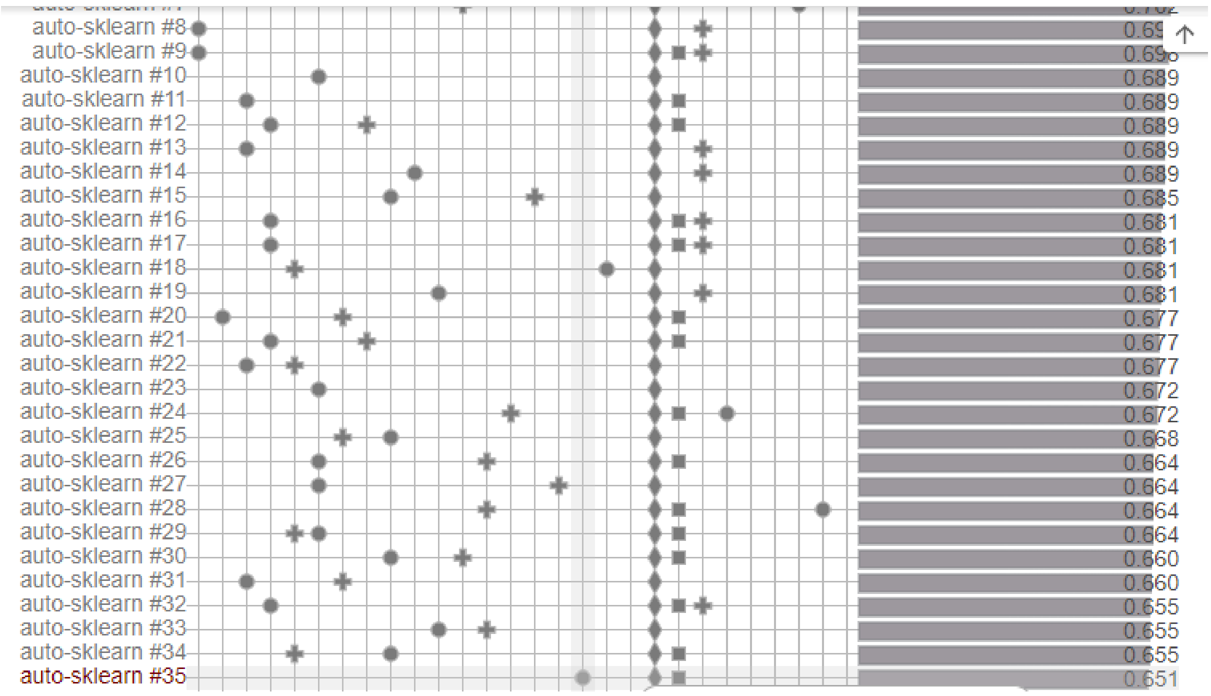
```
print(automl.sprint_statistics())
```

```
auto-sklearn results:
  Dataset name: 8570094d-a3d6-11ed-806f-0242ac1c000c
  Metric: accuracy
  Best validation score: 0.723404
  Number of target algorithm runs: 54
  Number of successful target algorithm runs: 52
  Number of crashed target algorithm runs: 1
  Number of target algorithms that exceeded the time limit: 1
  Number of target algorithms that exceeded the memory limit: 0
```

```
print(automl.leaderboard())
```

```
          rank  ensemble_weight                 type      cost  duration
model_id
28           1             0.10                  qda  0.276596  1.140786
20           2             0.06          extra_trees  0.285106  1.676613
4            3             0.06          extra_trees  0.293617  1.841954
9            4             0.02                  mlp  0.293617  1.680628
35           6             0.04                  lda  0.302128  0.961989
45           5             0.04                  lda  0.302128  1.091437
2            7             0.04        random_forest  0.310638  2.875540
22           9             0.06                  mlp  0.310638  4.662967
46           8             0.06                  qda  0.310638  1.095160
18          10             0.02    gradient_boosting  0.314894  1.418257
29          11             0.02  k_nearest_neighbors  0.319149  0.882213
38          12             0.22             adaboost  0.319149  1.078088
7           14             0.02          extra_trees  0.323404  2.222743
24          13             0.02                  mlp  0.323404  1.395840
21          15             0.04        random_forest  0.336170  2.682881
8           16             0.02                  mlp  0.344681  3.442860
44          17             0.08                  lda  0.357447  0.930164
51          18             0.02    gradient_boosting  0.357447  1.406636
14          19             0.06   passive_aggressive  0.361702  1.114763
```

```
pip install PipelineProfiler
import PipelineProfiler
profiler_data=PipelineProfiler.import_autosklearn(automl)
PipelineProfiler.plot_pipeline_matrix(profiler_data)
```



INTERPRETATION

Leaderboard shows 72% accuracy.